Efficient k-NN implementation for real-time detection of cough events in smartphones

Hoyos Barceló, Carlos; Monge-Álvarez, Jesús; Shakir, Muhammad Zeeshan; Alcaraz Calero, Jose M.; Casaseca, Juan Pablo

Published in:
IEEE Journal of Biomedical and Health Informatics

DOI:
10.1109/JBHI.2017.2768162

Published: 02/11/2017

Document Version
Peer reviewed version

Efficient *k*-NN Implementation for Real-Time Detection of Cough Events in Smartphones

Carlos Hoyos-Barceló, Jesús Monge-Álvarez, *Student Member*, IEEE, Muhammad Zeeshan Shakir, *Senior Member*, IEEE, Jose-María Alcaraz-Calero, *Senior Member*, IEEE, Pablo Casaseca-de-la-Higuera*, *Member, IEEE*,

***Abstract*— The potential of telemedicine in respiratory health care has not been completely unveiled in part due to the inexistence of reliable objective measurements of symptoms such as cough. Currently available cough detectors are uncomfortable and expensive at a time when generic smartphones can perform this task. However, two major challenges preclude smartphone-based cough detectors from effective deployment namely, the need to deal with noisy environments and computational cost. This paper focuses on the latter, since complex machine learning algorithms are too slow for real-time use and kill the battery in a few hours unless specific actions are taken. In this paper, we present a robust and efficient implementation of a smartphone-based cough detector. The audio signal acquired from the device's microphone is processed by computing local Hu moments as a robust feature set in the presence of background noise. We previously demonstrated that pairing Hu moments and a standard k-NN classifier achieved accurate cough detection at the expense of computation time. To speed-up k-NN search, many tree structures have been proposed. Our cough detector uses an improved vp-tree with optimized construction methods and a distance function that results in faster searches. We achieve 18x speed-up over classic vp-trees, and 560x over standard implementations of k-NN in state-of-the-art machine learning libraries, with classification accuracies over 93%, enabling real-time performance on low-end smartphones.**

Index Terms— Cough detection, Mhealth, Efficient Implementation, k-Nearest Neighbors, Vantage Point Trees.

Manuscript received 28/07/17; revised 27/09/17; accepted 22/10/17. Date of publication ZZZZ; date of current version 26/10/17. This work was supported by the Digital Health & Care Institute Scotland as part of the Factory Research Project SmartCough/MacMasters. The authors would like to acknowledge support from University of the West of Scotland for partially funding C. Hoyos-Barceló and J. Monge-Álvarez studentships. UWS acknowledges the financial support of NHS Research Scotland (NRS) through Edinburgh Clinical Research Facility. Acknowledgement is extended to Cancer Research UK for grant C59355/A22878.

**Asterisk indicates corresponding author*

C. Hoyos-Barceló, J. Monge-Álvarez, M. Z. Shakir, J.M. Alcaraz-Calero and P. Casaseca-de-la-Higuera* are with the Centre for Artificial Intelligence, Visual Communications and Networking (AVCN) School of Engineering and Computing, University of the West of Scotland, Paisley Campus, Paisley, PA1 2BE, United Kingdom. P. Casaseca-de-la-Higuera is also with the Laboratory of Image Processing (LPI), ETSI Telecomunicación, Universidad de Valladolid. 47011, Valladolid, Spain (email: pablo.casaseca@uws.ac.uk, casaseca@lpi.tel.uva.es).

I. Introduction

Cough constitutes the most recognizable symptom in respiratory disease and is associated with more than one hundred pathological conditions [1]. The economic burden of lung diseases has been estimated to be in the order of €96.4 billion per annum in the EU; plus another €283 billion in opportunity costs due to premature mortality [2]. Chronic Obstructive Pulmonary Disease (COPD) and asthma are the most prevalent, with a cost of €6,147–7,443 per patient per year [2]. Effective and objective tracking of symptoms such as unexpected increases in cough frequency would allow physicians to take preventive measures before a critical point is reached [3], thus avoiding hospitalization and significantly lowering the treatment costs.

The most common conditions involving chronic cough in non-smokers are asthma, gastro-esophageal reflux, and rhinitis [4]. Cough monitors can potentially aid physicians in their diagnostics, as different illnesses have different coughing patterns [5]. For instance, patients with bronchitis cough more than others [6], pneumonia tends to produce short and low-intensity coughs [7], and asthmatics may produce nocturnal coughs [8] [5]. Cough assessment methods rely on subjective questionnaires where data reported by the patient often has little connection with reality [9] [10]. Leconte et al. demonstrated that cough frequency correlated with perceived Quality of Life [11]. However, manually counting cough events is a tedious and error-prone task, prompting the need of automatic cough counters.

Automatic cough detection is an established field of research with a number of systems achieving high sensitivity and specificity [12] [13]. Most systems in the literature rely on pattern recognition engines based on features extracted from cough sounds and other biomedical signals (chest movements, electrocardiograms, etc.), with the latter acting as support for the microphone signal [14]. However, the vast majority of those systems can be considered as expensive and uncomfortable (i.e., non-wearable during daily activity) [9] solutions at a time when telehealth has moved towards generic readily available sensors. The recent advances in smart phone technology allow their use as intelligent cough monitoring systems as these are devices that patients can carry with themselves, and are able to capture sound and run general-purpose programs.

A major challenge when using a smartphone as a cough detector, resides in the need of dealing with noisy environments when processing the audio signal [15]. Most of the current systems identify the cough sounds using Mel Frequency Cepstral Coefficients (MFCC) obtained from the spectral analysis of the signal. However, this feature set fails to accurately detect cough sounds in noisy scenarios to an extent that currently available smartphone-based systems deactivate cough detection when the Signal to Noise Ratio (SNR) is perceived to be low [16].

Another significant challenge is battery consumption [15], since complex machine learning algorithms are prone to kill the smartphone battery in a few hours, whereas the user would prefer the app running seamlessly with no effect on the normal phone functionality. With no visible impact on daily activity, patients would be less conscious of the medicalization of their lives. The work in [9] solved this problem by using the phone only for data acquisition and off-loading the processing to an external server. Even in these conditions, the use of the phone connectivity and storage modules can constitute a significant burden in terms of consumption. An ideal cough monitor should withstand at least 24 hours of operation, but current classification systems are very computationally expensive, with several studies reporting battery issues [9] [17]. A call was made to develop more energy-efficient pattern recognition engines for mobile platforms [18].

Our preliminary work in [19] proposed using local Hu moments in the time-frequency domain as a robust feature set to carry out cough detection in noisy environments. Hu-Moments have been extensively used in image processing for object recognition [20], and were recently extended for speech emotion recognition [21]. By using them together with a k-NN (nearest neighbours, with k=1) classifier, we achieved sensitivity and specificity values of 94.17% and 92.16% respectively for SNR=15dB. This showed the synergy of local Hu-Moments with a widely used classifier in audio processing.

The advantage of k-NN over other classifiers is that it does not require a costly training phase, allowing the use of adaptive learning from the patient's input. Adaptive learning has shown superior performance over trained-once classifiers, with over 30% better recall rates [22] [23]. It also naturally solves the problem of sensor variability between phone devices. However, the main disadvantage is that queries are slow, as for every classification it must compute the distance of the new sample to all instances in the data set. In benchmarks, instance-based classifiers such as k-NN scored last in terms of speed and battery use [24]. Furthermore, accuracy in these classifiers is dependent of the size of the training set. So to get good detection rates, the database has to be large, thus making classification slower. Space and memory limitations become relevant as the entire database must be stored in the device and loaded into memory.

This paper presents a robust and efficient implementation of a smartphone-based cough detector. The audio signal acquired from the device's microphone is processed by computing local Hu moments as a robust feature set in the presence of background noise. These features feed an optimized k-NN classifier for final cough detection. A study on the effect of different optimizations of the k-NN algorithm is presented to propose a novel optimized version of the classifier. The proposed classifier speeds-up the nearest neighbor search using Vantage Point Trees where vantage points are selected over reduced datasets to optimize their construction. In addition, a simplified distance function is employed to speed-up calculations. Our results show that the proposed cough detector outperforms optimized implementations in state-of-the-art standard machine learning libraries while achieving high classification accuracy, thus enabling real-time performance on low-end smartphones.

II. OVERVIEW OF THE SYSTEM

An overview of the proposed system is sketched in
. The audio signal from the device's microphone is sampled at 8.82 kHz, a frequency shown appropriate for cough event detection [25]. After that, 50 ms windows with 25 ms shift are extracted for classification using a Kaiser window with $\beta = 3.5$. As a starting point, the power spectral density (PSD) for each window is extracted and normalized. Initial speed-up is performed at this stage by using optimized FFT algorithms and taking advantage of the 25 ms window overlap to perform only 50% of the required computations. A feature extraction block carries out the computation of local Hu moments as detailed in the Appendix to subsequently feed the pattern classification module where the efficient versions of the k-NN algorithm have been implemented.

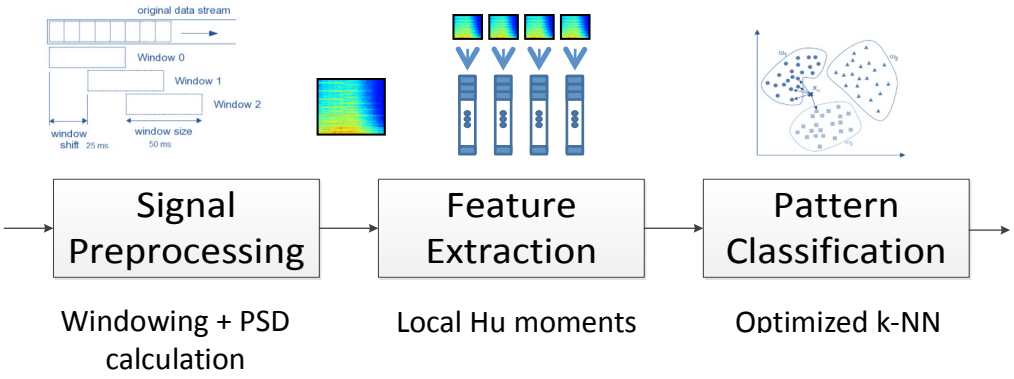

Fig. 1. Overview of the proposed cough detector

III. K-NN OPTIMIZATION

K-NN classifiers store a set of instances with pre-assigned classes, known as the training set, and classify new instances by assigning them the most common class among the k closest relatives in that set. To determine closeness, k-NN converts feature vectors to points in a multidimensional metric space; then, it measures the distance between points according to a distance function [26]. Fig. 2 illustrates classification using k-NN. The logic behind this classifier is that less similar instances are less likely to belong to the same class. In two-class classification problems, *k* is usually set to an odd number to prevent ties. If the classes in the training set are unbalanced, a large *k* skews the results towards the majority class, whereas low values of *k* are less stable against outliers. As a result, the optimum *k* is usually set experimentally.

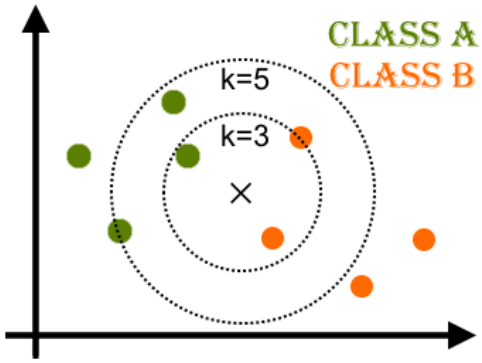

Fig. 2. Illustration of the classification process with k-NN, and how changing k impacts classification. In the example, a point located at X would be classified as B if k=3, or as A if k=5.

A. Metric trees

Basic (naïve) k-NN search has ***O(n)*** cost because all points in the training set are compared against the query, even if they are far apart in the metric space. *Metric trees* [27] are tree-like data structures that logically split the metric space in sub-regions, with each child node containing about half the points of its parent. This partitioning allows to perform binary search on the tree, which brings down the cost of queries to ***O(log n)***. The following subsections describe three popular structures namely, KD-trees, Ball-trees, and (Vantage Point) VP-trees.

1) KD-trees

KD-trees partition the geometric space using hyperplanes as shown in Fig. 3. Non-leaf nodes have two child nodes, and define hyperplanes that split the metric space alongside one dimension. Points below a certain threshold value go to the left child, and the rest go to the right child. The thresholds can be chosen in several ways. For our experiments, we implemented the *Median of Widest Dimension* construction method described by Kibriya [28]. Branching stops whenever a child node contains equal or fewer points than a threshold referred to as *bucket size*. Lower values of this parameter result in more nodes and smaller regions. The performance of KD-Trees degrades as the dimensions increase, becoming slower than linear search [28].

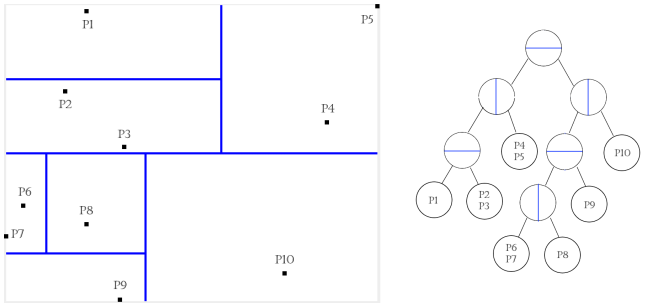

Fig. 3. Illustration of a kd-tree created by partitioning the metric space with hyperplanes.

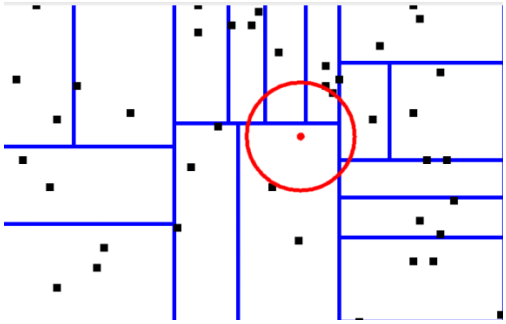

Fig. 4. Search is limited to those regions that overlap with the ball centered at the query point with radius equal to the distance to the current best k-NN.

Queries in KD-Trees start with a depth-first search to the leaf node that would contain the query point (see Fig. 4). Then, it calculates the distance to all the points within this node. The distance to the best k-Nearest Neighbor found so far will be used as the radius of a hyperball centered at the query point. If the hyperball is within the margins of the region, the algorithm stops; otherwise, the algorithm goes back up one level in the tree and checks for overlaps between the ball and the other child nodes, visiting them and updating the ball radius as better neighbors are found. The backtracking ends once the root node has been explored.

2) Ball-trees

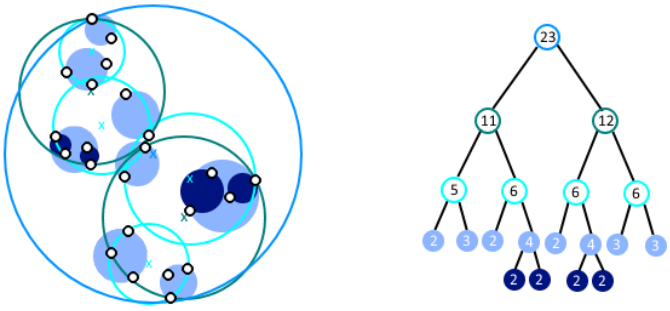

Fig. 5. Illustration of a ball-tree created by clustering points inside minimal hyper-balls

As illustrated in Fig. 5, ball-trees partition the space through the use of minimal hyper-balls that completely contain all of their children, while overlapping the least with other sibling spheres. Unlike KD-Trees that are based on coordinate values, ball-trees define regions in terms of relative distance to the

centroids of child nodes. Queries exploit the **reverse triangle inequality** property of metric spaces to skip the evaluation of nodes that cannot contain points closer than the current best candidate. Ball-trees are hard to construct optimally, and also become less efficient than linear search for high dimensions [28]. For our experiments, we implement the top-down Ball-tree construction method proposed by Omohundro [29]. Searches first visit the hyper-balls whose centroid is closest to the query point, and also those that overlap with the ball centered at the query, with radius equal to the distance to the best k-NN found so far.

3) VP-trees

Yianilos et al. [30] proposed *Vantage Point trees* (vp-tree), a structure similar to ball-trees, except that the center of the hyper-ball is always a point in the node, termed the *vantage point*. To split the space, the radius μ of this hyperball is set to a value that encompasses half of the points of the parent node, leaving the rest outside (see Fig. 6). The left child will contain all the inner points, whereas the right outer child will contain the rest.

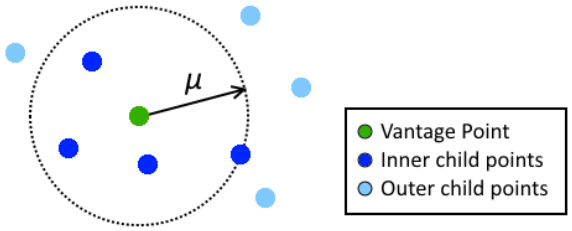

Fig. 6. A non-leaf node in a vp-tree is defined by a vantage point and a radius μ that splits the points in 2 disjoint sets.

Search commences by comparing the distance of the query point to the vantage points of non-leaf nodes, starting from the root node. If the distance is lower than μ, the inner child is explored, otherwise the outer child. Once a leaf-node is reached, it measures the distance of the query to all the points contained in it, storing the distance d_k to the current best k-NN. This value will define the radius of a hyperball with the query point at its centre. If the hyperball does not intersect other regions, the search ends; otherwise, we go up the tree and check for regions that overlap with the ball, updating the values as better candidates are found, until the root node is evaluated. To check if there is an overlap, we measure the distance τ of the query to each parent vantage point, with overlaps occurring if $d_k \geq |\tau - \mu|$ (see Fig. 7).

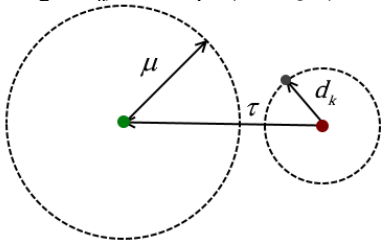

Fig. 7. In this example, the query does not overlap with the hyperball of the vantage point, so points inside that ball can be safely ignored.

B. Distance functions

Distance functions are used to calculate the relative closeness between two instances or points in the metric space. They can have impact both in classification accuracy and efficiency, especially for the latter, when complex computations are involved in the calculations. The most commonly used metric is *Euclidean distance* (L_2-norm of the difference vector), which computes the distance between points $p = (p_1, p_2,..., p_n)$ and $q = (q_1, q_2,..., q_n) \in \mathbb{R}^n$, according to:

$$Euclidean(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \quad (1)$$

As the square root is an expensive operation, we considered two other distance functions that are simpler to compute, paying a small accuracy penalty in exchange for faster classification times. *Manhattan distance* (2), for example, is the sum of the absolute differences for each coordinate (L_1-norm of the difference vector).

$$Manhattan(p,q) = \sum_{i=1}^{n}|q_i - p_i| \quad (2)$$

We also consider the squared L_2-norm of the difference vector, which is denoted as $Euclidean^2$ in (3). In this case, we avoid the calculation of the square root for efficiency. This function is a *pseudo-metric*, as it does not always satisfy the *triangle inequality* on which many metric trees are based. It is thus worth noting that using it can lead to accuracy loss when building Ball-trees or VP-trees.

$$Euclidean^2(p,q) = \sum_{i=1}^{n}(q_i - p_i)^2 \quad (3)$$

C. Database pruning

One common way to improve search times in instance-based classifiers is to reduce the size of the database. Hart's algorithm, also known as Condensed Nearest Neighbour, is an algorithm for data reduction that aims to remove redundant instances of the database [31]. In our experiments, we evaluated the different implementations using a pruned training set constructed with Hart's algorithm. The size of the pruned database was 17.5% of the original one. Even though this can lead to significant savings in computation time, the impact on the classification accuracy can be major.

D. Feature normalization

Direct distance calculation from non-normalized feature vectors can result in one dimension dominating over the others. For example, if a feature f_1 ranges from [0,100] and f_2 ranges [0,1], f_1 will have more impact in the returned distance even if f_2 is a better discriminator. To avoid this, a common practice in machine learning is to normalize all values in the training set so every dimension has a [0,1] range, scaling the feature vector of the query points accordingly for proper comparison.

IV. EXPERIMENTS

A. Experimental setup

We implemented different versions of the k-NN classifier. The baseline one used linear search (no optimization) and was used as a reference for comparison with the optimized version, and as first testbed to assess the impact of different distance metrics, normalized/non-normalized feature sets, and pruning of the database using Hart's "Condensed NN" algorithm [31]. Due to the lower performance in cough detection for $k>1$ [19], we based our experiments in 1-NN. The optimized versions respectively used VP-Trees, KD-Trees and Ball-Trees structures for fast search. All the trees perform *partial sort* to find the *k*-NN. Once a decision on the structure was made, we evaluated different tree construction strategies and the impact of parameters such as the *bucket size* in the search.

As performance metrics, we used sensitivity and specificity, respectively computed as the ratios *Detected cough windows/Total cough windows*, and *Identified non-cough windows/Total non-cough windows*. Overall accuracy was also computed as the pooled average of those values weighted with the respective cough/non-cough frequencies. In terms of efficiency, we used the processing time and calculated the relative speed-up compared with a standard base implementation. For this standard implementation, we chose the widely used Machine Learning library Weka. A freely available version of the library for the Android platform can be found in [32]. Weka's naïve (non-optimized) algorithm reads the training set and normalizes it. Then, for each query, it computes the distances from all points in the database to the query point, sorts the list, and returns the *k* closest neighbours to the query. Weka also features optimized versions of k-NN using KD-Trees and Ball-Trees. We used these versions for comparison purposes.

The smartphone used for the tests is a **Sony Xperia Z2, running on Android 5.1.1.** The computed times obtained in this study correspond to an average of five measures taken for each option.

B. Cough database

The cough database used in evaluation consists of a library of sounds recorded at 44.1 kHz with a precision of 16 bit, then down-sampled to 8820Hz and grouped in windows of 441 samples (50ms) with 50% overlap. The database contains patient and voluntary cough events and other relevant audio events such as speech, laugh, throat clearing, etc. The events were contaminated with background noise from various sources, such as environmental noise from roadworks, street with vehicles passing by, a crowd, etc. The sounds in the database have a total duration of 1245s. The database was split into three sets: training (60%), validation (10%), and testing (30%). The training set has a total of 30139 windows, whereas the test set has 15069. We kept the same ratio of cough/no cough windows for all these sets, at 18.57%, and we included samples with SNR values between -15 and 0 dB. To deal with the unbalance between classes, we performed importance sampling using the Distinct-Borderline2 Synthetic Minority Oversampling technique (DB2SMOTE) and a cost matrix as in [19]. Only the feature vector and label for each window are stored in the device. The whole database is contained in a 9.8MB file stored in an efficient binary format.

C. Results and discussion

1) Impact of distance metric, normalization, and size of the database in baseline model

Fig. 8 shows a scatter diagram of the computation time vs. overall accuracy for our linear search (non-optimized) k-NN implementation. The relevant parameters for this analysis are been coded as follows:

1) Normalized/Unscaled features (color coded);
2) Different distance metrics (shape coded);
3) Full/Pruned (17.5%) database (transparency coded).

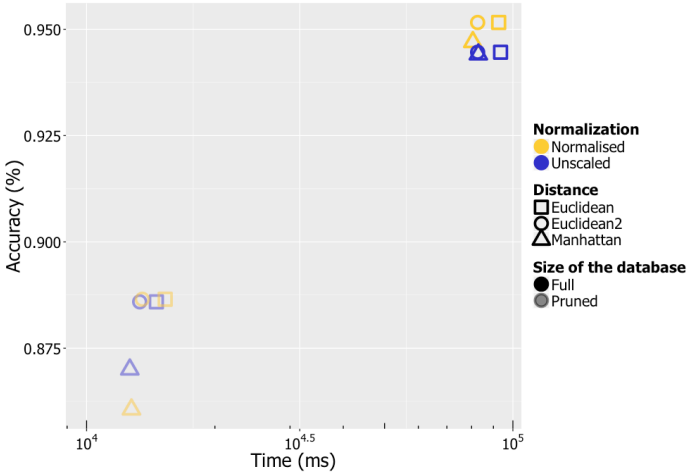

Fig. 8. Computation time vs. overall accuracy for non-optimized k-NN classifier with: 1) Normalized/Unscaled features (color coded); 2) Different distance metrics (shape coded); 3) Full/Pruned (17.5%) database (transparent).

We can conclude on the following from the figure:

- The Euclidean^2 distance function achieves the same results as Euclidean in terms of accuracy, with lower computation times. Manhattan distance did not provide speed advantages over Euclidean^2, while being itself less accurate. Thus, for further analysis, we will keep the comparison between Euclidean and Euclidean^2, setting Manhattan aside for the sake of simplicity.
- Normalization has a positive impact on the accuracy of the classifier for non-pruned databases. This can be used later on to improve accuracy of the finally selected optimized classifier.
- We also observed that the use of Hart's algorithm reduces the accuracy of the classifiers below acceptable levels, even though the computational savings are also significant. This is especially relevant in terms of sensitivity, as can be seen in Fig. 9, where the scatter plot shows sensitivity vs. specificity.

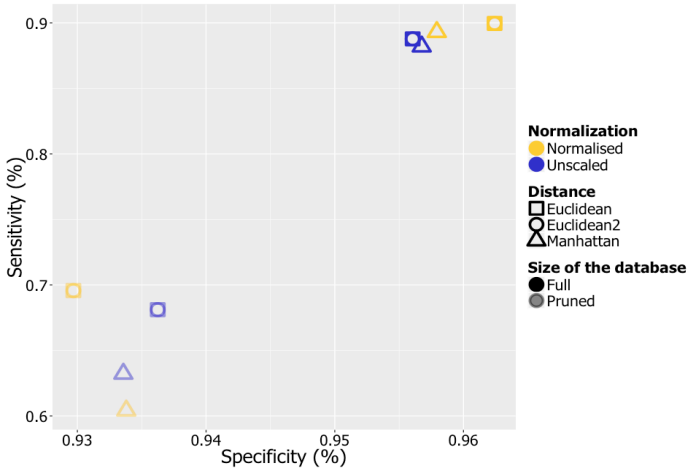

Fig. 9. Sensitivity vs. Specificity for linear search k-NN classifier with: 1) Normalized/Unscaled features (color coded); 2) Different distance metrics (shape coded); 3) Full/Pruned (17.5%) database (transparency coded).

2) Efficiency improvement from metric trees

Fig. 10 presents a scatter diagram of the computation time vs. overall accuracy for different optimizations (color coded) of the k-NN algorithm using metric trees. The standard Weka library implementations are presented for the sake of comparison. We have used transparency coding in the figure to specify which implementation (Weka or ours) was used.

Weka's algorithm performs some steps that are unnecessary in 1-NN search, like sorting the array of distances. As this operation has a substantial $O(n \cdot log(n))$ cost, we also include our own 1-NN linear-search algorithm for a more apt comparison. The distance metrics are specified in the figure using shape coding as before.

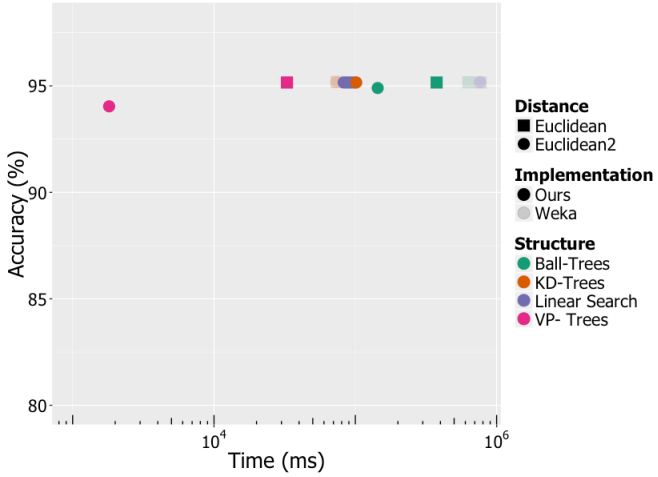

Fig. 10. Computation time vs. overall accuracy for different k-NN implementations using metric trees. Baseline non-optimized linear versions are also presented for comparison. Parameters are encoded as follows. 1) Type of structure, metric trees (KD, VP and Ball), and non-optimized version (color coded); 2) Distance metric (shape coded); 3) Implementation, ours or standard library Weka (transparency coded).

The following conclusions can be extracted from the figure:

- Our linear-search 1-NN classifier is one order of magnitude faster than Weka's implementation, remarking the advantages of skipping the sort step in the 1-NN case.
- VP-Trees are the fastest structure, and present an unforeseen synergy with Euclidean^2 distance. The resulting classifier is one order of magnitude faster than the one using Euclidean, and three orders of magnitude faster than the Weka's base implementation.
- The other structures, particularly ball-trees, yield unacceptable computation times, even higher than linear search. A possible explanation for this is the "curse of dimensionality", which results in poor performance for KD-trees and Ball-trees for dimensions larger than 10 [33].
- We did not expect Euclidean^2 to be a good distance function for VP-trees, as it violates the triangle inequality on which this structure is based, so we sought an explanation. Ting Liu et al. reported that metric trees typically find a very good nearest neighbor candidate in the first moves, then spend up to 95% of the time verifying that they got it right [34]. From this, we infer that Euclidean^2 results in less overlaps with surrounding regions, speeding up the queries.

We rank the classifiers in terms of **speed-up**, which is computed as the time required by Weka to classify all instances, divided by the time it takes the other classifiers to do the same.

- The default naïve k-NN algorithm available in the Weka library normalizes the dataset and takes an average 769,728ms to classify the 15069 test samples with 95.16% accuracy. At roughly 51ms per sample, the performance is unacceptably slow for real-time use, as new windows are generated every 25ms, and the system must also acquire the audio signal, calculate spectral features and compute the Hu Moments.
- Our linear-search 1-NN takes 82663ms when using Euclidean^2, which is **9.31x** faster than Weka.

The following experiment shows the impact of carrying out feature normalization in the preferred VP-tree structure. Table I shows computation time and overall accuracy for the VP-Trees implementations using both Euclidean and Euclidean^2 distances. The performance obtained with Weka implementation using both the full database and a pruned one is also presented for the sake of comparison. Table II summarizes the speedups achieved when using our VP-Trees implementations compared to Weka. We can extract the following conclusions from the tables:

- Using non-normalized versions of the feature set significantly improves efficiency at a low accuracy cost.
- The speedup obtained from VP-Trees using Euclidean^2 distance is very significant (**425x** and **559x** for normalized and unscaled features respectively) while keeping high classification performance. The speed-up over classic VP-Trees using Euclidean distance is **18.12x** for normalized features and **13.23x** for unscaled.
- Even in comparison with a pruned database, the speedup is highly valuable (**71x** and **94x** for normalized and unscaled features respectively), especially considering that Weka's classification performance drains in this case.

TABLE I
COMPUTATION TIME VS. OVERALL ACCURACY VP-TREES IMPLEMENTATIONS COMPARED TO WEKA (FULL AND PRUNED DATABASE).

Accuracy (%)	Time (ms)	Implementation	Distance	Normalized
95.16	769728	Weka (full DB)	Euclidean	Yes
95.16	762757	Weka (full DB)	Euclidean^2	Yes
87.69	129239	Weka (pruned DB)	Euclidean	Yes
87.60	124239	Weka (pruned DB)	Euclidean^2	Yes
95.16	32818	VP- Trees	Euclidean	Yes
94.04	1811	VP- Trees	Euclidean^2	Yes
93.24	1377	VP- Trees	Euclidean^2	No
94.46	18226	VP- Trees	Euclidean	No

TABLE II
SPEEDUP OF VP-TREES IMPLEMENTATIONS COMPARED TO WEKA WITH FULL AND PRUNED DATABASE

Distance	Normalization	Speedup		Accuracy (%)
		Full Database	Pruned Database	
Euclidean	Normalized	23.45	3.94	**95.16%**
Euclidean	Unscaled	**42.23**	**7.09**	94.46%
Euclidean^2	Normalized	425.03	71.36	**94.04%**
Euclidean^2	Unscaled	**558.99**	**93.86**	93.24%

According to the results presented so far, we select the VP-Tree structure together with Euclidean^2 distance as our baseline optimized implementation for the k-NN algorithm. The following experiments explore further optimizations of this structure for the final proposal.

D. Optimizing VP-Tree construction for Euclidean^2 distance

1) Selection of vantage points

The efficiency of k-NN search in VP-Trees depends on the number of regions that overlap with the query's hyperball. When creating VP-Trees, how the vantage points are picked up has a significant impact on the odds of overlap. Most implementations pick the vantage points at random to keep the cost of building the tree $O(n \cdot log(n))$, but Yanilos et al. described a method for picking vantage points that minimizes overlaps [30]. They found that vantage points that maximized the second moment of their distances from all the other points, which are notably located in the corners of the space (see Fig. 11), resulted in faster searches [30].

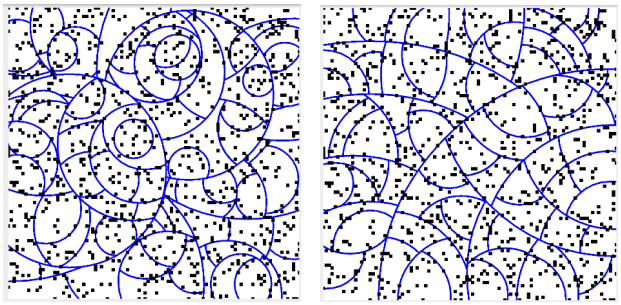

Fig. 11. Two VP-Trees on the same set of points. The left one assigns vantage points at random while the right one picks points at the corners of space.

Building an *optimal* vp-tree according to Yanilos's criteria has a prohibitive $O(n^2 \cdot logn)$ cost, so we implement instead an *approximation*; for nodes with over 1000 points, we only evaluate 10% of candidate vantage points against 25% of the remaining points in the node. As tree creation has a random component, this naturally results in random performance of the classifier. To ensure deterministic behavior between sessions of the app, we store the generated tree in a binary-efficient file and load it at start-up. We found out that searches on trees loaded from memory were up to 60% faster than on newly-constructed trees, which we attribute to better cache locality.

2) Effect of bucket size

The bucket size chosen at VP-Tree construction controls the size and number of regions in which the space is subdivided. Higher values make searching each region slower, but since it also reduces the number of overlapped regions, there is a range that results in faster queries.

A larger bucket size improves accuracy. Wrong results when using Euclidean^2 are caused by the algorithm not visiting the region that contains the true k-NN, and larger bucket sizes increase the odds that the nearest neighbour will be found on the first leaf nodes visited.

We tested bucket sizes ranging from 1 to 500 points. Fig.12. shows the accuracy obtained using both tree construction methods. For reference, we also include the obtained accuracy using Euclidean distance as an upper bound, and for the lower bound we used the so-called "Defeatist" search, which stops after exploring the first leaf node [34]. The following conclusions can be extracted from the figure.

- Even though VP-trees created using Yanilos's criteria were notably faster, trees using this construction method did not pair well with Euclidean^2 distance, reducing accuracy and sensitivity by a significant amount. We therefore chose the random method as it performed better and was fast enough for our real-time needs. However, the use of Yanilos' construction method can be an option for slower devices.
- For random method, accuracy increases proportionally to bucket size until the value of 20 is reached, then stops improving. This bucket size was thus finally selected, as it was also fast enough for real-time computation.

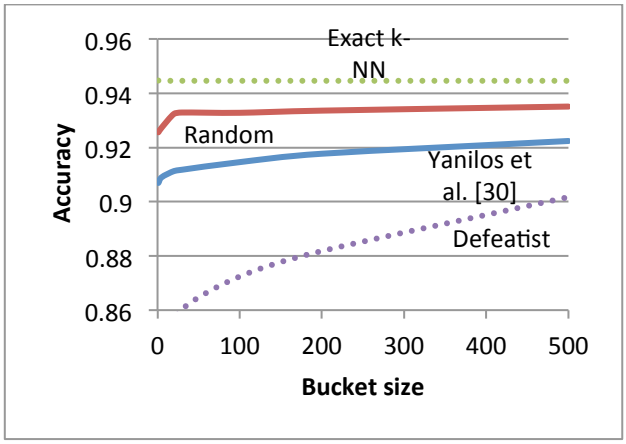

Fig.12 Overall Accuracy vs. bucket size for different Vantage point pick-up strategies in VP-Trees construction. Exact k-NN and defeatist strategy are also presented as upper and lower bounds for comparison.

3) Multiple VP-trees

A final optimization to improve accuracy at a expense of higher computation times would rely on the use of multiple VP-trees to finally pick the one providing the closest nearest neighbor. Fig. 13 shows the visual projection in 2D space of VP-Trees constructed with different bucket sizes. Spots that violate the triangle inequality when using Euclidean^2 distance are marked in yellow. There are two points of note here:

- For bucket sizes above 20, yellow points become confined to the boundaries of the sub-regions. Errors are exclusively dependent on tree structure.
- By picking vantage points at random, the final result is that two VP-Trees will have totally different boundaries despite containing the same points. In other words, it is unlikely that errors appear in the same place for both trees. So, using multiple trees and selecting the one with the closest nearest neighbour will yield higher accuracies.

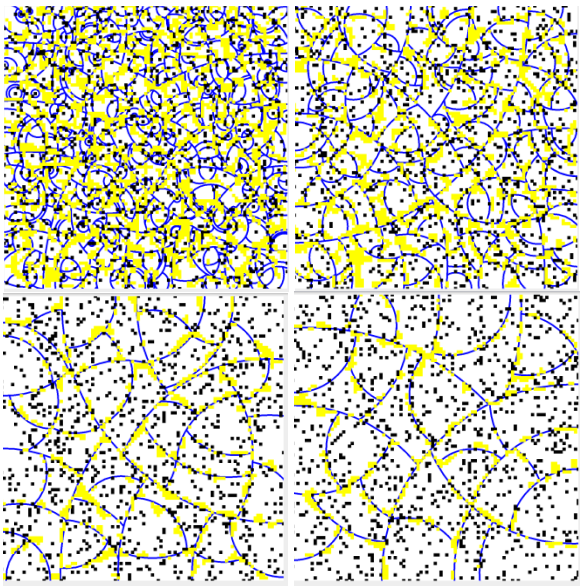

Fig. 13 VP-Trees with bucket sizes 2, 8, 30, and 50 (clockwise, starting from top-left). Points in yellow mark points in space in which Euclidean^2 distance returns the wrong k-NN.

We test k-NN search on several Random VP-trees using Euclidean^2, updating the k-NN if a later vp-tree finds a better candidate. The results using one, two, or three vp-trees are displayed on Fig. 14. Using two vp-trees significantly improves accuracy, whereas using 3 yields accuracies in line with the exact k-NN classifier. A 3-vp-tree classifier using normalized features achieves **95.07%** accuracy in 5824ms, which is still a speed-up of **132.16x** over Weka's implementation.

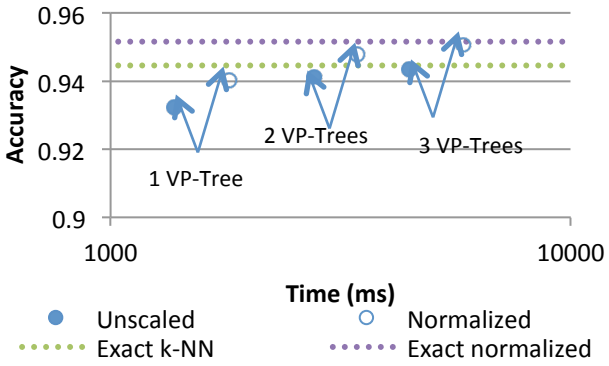

Fig. 14 Accuracy vs. processing time of multi-VP-tree classifier with 1, 2, or 3 VP-trees, for both normalized and unscaled features. Baseline accuracies for exact k-NN classification are presented for reference.

This solution has an increased memory cost, as it requires to store extra VP-tree structures. However, this cost is minimized since all tree structures share the same data points.

V. Related Work

Since the introduction of VP-trees, many studies have used them to speed up k-NN search. Most research focus, however, has been on improving efficiency of k-NN search in high dimensions, where performance degrades to linear search. Approximate k-NN algorithms typically project points to a lower-dimensional space to handle the curse of dimensionality, and then carry the search on a standard VP-tree [35].

Because VP-trees are based on the triangle inequality, researchers rarely test them with distance functions that do not satisfy it. In cases where they do, they modify the pruning function to enforce correctness [36]. When this is done for the Euclidean^2 metric, it results on exact k-NN that is just slightly faster than Euclidean [37].

Ting Liu et al proposed a refinement to Defeatist Search that duplicates points and assigns them to more than one leaf node [34]. However, implementation is tricky, and the idea assumes the tree will remain static, which is not the case of our system.

Our work shows that a Euclidean^2 metric with no correctness enforcement is a simpler way to improve efficiency of VP-trees, obtaining high speed-ups over standard VP-tree search even at low dimensions. We also identified additional strategies to improve accuracy or speed to meet the performance goals of the target platform.

VI. Conclusion and Future Lines

This paper presents a robust and efficient implementation of a smartphone-based cough detector. Previous studies

discarded the k-NN classifier as unacceptably slow and resource hungry for real-time processing in smartphones, while others had pointed out that the classification stage was the costliest part of the system, taking 31–44.9% of the CPU time, and being up to 8 times slower than the feature extraction module [38] [39]. The standard (non-optimized) Feature calculation of Hu Moments takes approximately 22 ms per window when implemented on the smartphone used in our experiments. Adding the 51ms to classify each window with Weka's implementation prevented real-time detection, as new windows are generated every 25 ms.

We unveiled an interesting synergy between VP-trees and Euclidean^2 distance that results in a k-NN classifier that is fast and able to perform real-time detection on smartphones, without needing GPU acceleration or server off-loading. Our implementation classifies a new window in less than 1% of the CPU time spent computing the feature vector, performing the overall computation in less than 23ms.

Our next goal will be to improve the efficiency of the rest of the modules of the system, to optimize database size, detect and correct false positives, and maximize the battery life of the device. To improve the accuracy in real life-scenarios, we plan to include a self-training module for personalized calibration. Such a module would increase the computation cost, as it involves updating the indexing tree. For efficiency reasons, it should not continuously run all the time, but rather would only be activated at user request for short-period sessions.

APPENDIX. COMPUTATION OF LOCAL HU MOMENTS

Local Hu moments are computed for each signal window *w[n], n=1,…N,* extracted from the acquired audio signal. As mentioned in Section II, a normalized PSD is computed *PSD[l], l=1,…Nfft*, where *Nfft=4096*. The next step calculates the logarithm of the spectral energies for every window in a series of bands defined by a filterbank in the Mel scale:

$$E_k(m) = \log\left(\sum_{f=f_{\min}}^{f_{mas}} PSD_k[f]\cdot H_m[f]\right) \quad 0 \le m < M \tag{1}$$

where *k* refers to the *k-th* window and *m* denotes each filter within the filterbank. The *f* values correspond to *Nfft* discrete frequencies in the range [f_{min}, f_{max}], where $f_{\min}$ and $f_{\max}$ are 0 and 2 kHz, respectively. The filterbank in the Mel scale is defined as:

$$H_m(f) = \begin{cases} 0, & f < C(m-1) \\ \dfrac{2\cdot(f - C(m-1))}{(C(m+1)-C(m-1))(C(m)-C(m-1))}, & C(m-1) \le f < C(m) \\ \dfrac{2\cdot(C(m+1)-f)}{(C(m+1)-C(m-1))(C(m+1)-C(m))}, & C(m) \le f < C(m+1) \\ 0, & f \ge C(m+1) \end{cases} \tag{2}$$

$C(m)$ $0 \le m \le M$ are the central frequencies for each filter in the filterbank [Hz], uniformly spaced between $f_{\min}$ and $f_{\max}$ in the Mel scale. Conversion from natural frequencies to the Mel scale and vice versa is performed as:

$$f[Mel] = 2595\cdot\log_{10}(1 + f[Hz]/700) \tag{3}$$

$$f[Hz] = 700\cdot\left(10^{f[Mel]/2595} - 1\right) \tag{4}$$

The total number of employed filters is $M = 75$. Consequently, after performing this step for all the signal windows, a $(K \times (M-1))$ matrix was obtained, with K the number of signal windows.

Next, the local Hu moments of the energy matrix E are calculated by dividing E into $(K \times ((M/w)-1))$ blocks B_{ij}, with w the block size. In our calculation, we used $w = 5$ as in [21]

$$B_{ij} = \begin{pmatrix} E_i(w\cdot j) & \cdots & E_i(w\cdot j + w - 1) \\ \vdots & \ddots & \vdots \\ E_{i+w-1}(w\cdot j) & \cdots & E_{i+w-1}(w\cdot j + w - 1) \end{pmatrix} \tag{5}$$

$i = 1\ldots K \quad j = 1\ldots((M/w)-1)$

The latest $(w-1)$ blocks, corresponding to $i = K - w + 2, \ldots, K$, are padded with zeros up to the size $(w \times w)$. We got the first invariant moment θ of each B_{ij} as:

$$\theta = \eta(p=2,q=0) + \eta(p=0,q=2) \tag{6}$$

$$\eta(p,q) = \frac{\mu(p,q)}{(\mu(0,0))^{\rho}}, \quad \rho = (p+q+2)/2 \tag{7}$$

$$\mu(p,q) = \sum_{u=1}^{w}\sum_{v=1}^{w}(u-\bar{u})^p\cdot(v-\bar{v})^q\cdot g(u,v) \tag{8}$$

$$g(u,v) = B_{ij}(u,v) \quad p,q = 0,1,2,\ldots$$

In (8), $\bar{u}$ and $\bar{v}$ are $\bar{u} = \varphi(p=1,q=0)/\varphi(p=0,q=0)$ and $\bar{v} = \varphi(p=0,q=1)/\varphi(p=0,q=0)$, with:

$$\varphi(p,q) = \sum_{u=1}^{w}\sum_{v=1}^{w}u^p\cdot v^p\cdot g(u,v) \tag{9}$$

All θ are used to build a real $(K \times ((M/w)-1))$ matrix, Q. To conclude, the discrete cosine transform (DCT) is computed for each row in Q and coefficients 2-14 are finally kept. The result is a $(K \times 13)$ matrix TQ, being the rows of this matrix the Hu moments for each window in the signal.

ACKNOWLEDGMENT

The authors would like to thank the SmartCough clinical team at the University of Edinburgh. Dr. Lucy McCloughan, Prof. Brian McKinstry, Dr. Hillary Pinnock, and Dr. Roberto Rabinovich, who provided valuable support in clinical matters.

Additional thanks are given to Lorna Stevenson, Dave Bertin, and Jill Adams, from Chest Heart and Stroke Scotland, for setting up the patient panel for the SmartCough project.

REFERENCES

[1] G. A. Fontana, and J. Widdicombe, "What is cough and what should be measured?," *Pulm Pharmacol Ther,* vol. 20, no. 4, pp. 307-312, 2007.

[2] European Respiratory Society (ERS), "The economic burden of lung disease," in *European Lung White Book*, Available: Chapter 2. pp. 16-27, 2005.

[3] J. Smith and A. Woodcock, "Cough and its importance in COPD," *International Journal of Chronic Obstructive Pulmonary Disease,* vol. 1, no. 3, pp. 305-314, 2006.

[4] R. S. Irwin, “Assessing Cough Severity and Efficacy of Therapy in Clinical Research: ACCP Evidence-Based Clinical Practice Guidelines,” *Chest,* vol. 129, no. 1, pp. 232s-237s, 2006.

[5] S. Ranjani, V. Santhiya, and A. Jayapreetha, "A Real Time Cough Monitor for Classification of Various Pulmonary Diseases," in *Third International Conference on Emerging Applications of Information Technology (EAIT)*, pp. 102-105, Kolkata, *Nov. 29*, 2012.

[6] J. Hsu *et al.*, “Coughing frequency in patients with persistent cough: assessment using a 24 hour ambulatory recorder,” *European Respiratory Journal,* vol. 7, pp. 1246–1253, 1994.

[7] Y. A. Amrulloh, U. R. Abeyratne, V. Swarnkar, R. Triasih and A. Setyati, “Automatic cough segmentation from non-contact sound recordings in pediatric wards,” *Biomedical Signal Processing and Control,* vol. 21, no. August, pp. 126-136, 2015.

[8] J. Smith and A. Woodcock, “New Developments in the Objective Assessment of Cough,” *Lung,* vol. 186, no. 1, pp. 48-54, 2008.

[9] E. C. Larson, T. Lee, S. Liu, M. Rosenfeld and S. N. Patel, “Accurate and Privacy Preserving Cough Sensing using a Low-Cost Microphone,” in *UbiComp '11,* pp. 375-384, Beijing, *Sep. 17-21*, 2011.

[10] A. M. Li *et al.*, “Cough frequency in children with mild asthma correlates with sputum neutrophil count,” *Thorax,* vol. 61, pp. 747-750, 2006.

[11] S. Leconte, G. Liistr, P. Lebecque and J.-M. Degryse, “The objective assessment of cough frequency: accuracy of the LR102 device,” *Cough,* vol. 7, no. 11, pp. 1-8, 2011.

[12] J. Amoh and K. Odame, “DeepCough: A Deep Convolutional Neural Network in A Wearable Cough Detection System,” in *IEEE Proceedings of Biomedical Circuits and Systems Conference (BioCas 2015)*, pp. 438-441, Atlanta, 2015.

[13] S.-H. Shin, T. Hashimoto and S. Hatano, “Automatic Detection System for Cough Sounds as a Symptom of Abnormal Health Condition,” *IEEE Transactions on Information Technology in Biomedicine,* vol. 13, no. 4, pp. 486-493, 2009.

[14] T. Drugman *et al.*, “Objective Study of Sensor Relevance for Automatic Cough Detection,” *Journal of Latex Class Files,* vol. VI, no. 1, pp. 1-8, 2007.

[15] E. Agu *et al.*, "The smartphone as a medical device: Assessing enablers, benefits and challenges," in *2013 IEEE International Conference on Sensing, Communications and Networking (SECON)*, pp. 76-80, New Orleans, LA, USA, 2013.

[16] S. Larson *et al.*, “Validation of an Automated Cough Detection Algorithm for Tracking Recovery of Pulmonary Tuberculosis Patients,” *Plos one,* vol. 7, no. 10, pp. 1-10, 2012.

[17] M. Sterling, H. Rhee and M. Bocko, “Automated Cough Assessment on a Mobile Platform,” *Journal of Medical Engineering,* vol. 2014, pp. 1-9, 2014.

[18] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez and S. Pastrana, “Power-aware anomaly detection in smartphones: An analysis of on-platform versus externalized operation,” *Pervasive and Mobile Computing,* vol. 18, pp. 137-151, 2015.

[19] J. Monge-Álvarez, C. Hoyos-Barceló, P. Lesso, J. Escudero, K. Dahal and P. Casaseca-de-la-Higuera, “Effect of Importance Sampling on Robust Segmentation of Audio-cough Events in Noisy Environments,” in *38th Annual International Conference of the IEEE EMBC*, pp. 3740-3744, Orlando, Florida, 2016.

[20] L. Zhang, F. Xiang, J. Pu and Z. Zhang, “Application of improved HU moments in object recognition,” in *Proc. Int. Conf. on Automotion and Logistics*, pp. 554-558, Zhengzhou, 2012.

[21] Y. Sun, G. Wen and J. Wang, “Weighted spectral features based on local Hu moments for speech emotion recognition,” *Biomed Signal Process Control,* vol. 18, no. April, pp. 80-90, 2015.

[22] K. Yatani and K. N. Truon, “BodyScope: A wearable acoustic sensor for Activity Recognition,” in *UbiComp '12*, pp. 341-350, Pittsburgh, *Sep. 5-8*, 2012.

[23] V. Könönen, J. Mäntyjärvi, H. Similä, J. Pärkkä and M. Ermes, “Automatic feature selection for context recognition in mobile devices,” *Pervasive and Mobile Computing,* vol. 6, pp. 181-197, 2010.

[24] R. E. Guinness, “Beyond Where to How: A Machine Learning Approach for Sensing Mobility Contexts Using Smartphone Sensors,” *Sensors 15,* pp. 9962-9985, 2015.

[25] P. Casaseca-de-la-Higuera *et al.*, "Effect of downsampling and compressive sensing on audio-based continuous cough monitoring," in *Proccedings of the IEEE Annual International Conference of the Engineering in Medicine and Biology Society*, pp. 6231-6235, Milan, Italy, 2015.

[26] N. S. Altman, “An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression,” *The American Statistician,* vol. 46, no. 3, pp. 175-185, 1992.

[27] T. Liu, “Fast Nonparametric Machine Learning Algorithms for High-dimensional Massive Data and Applications,” School of Computer Science Carnegie Mellon University, Pittsburgh, 2006.

[28] A. M. Kibriya, Fast Algorithms for Nearest Neighbour Search, Hamilton, New Zealand: The University of Waikato, 2007.

[29] S. M. Omohundro, “Five balltree construction algorithms,” International Computer Science Institute of Berkeley, California, 1989.

[30] P. N. Yianilos, “Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces,” in *Forth ACM/SIGACT-SIAM Conference on Discrete Algorithms (SODA)*, pp. 311-321, Austin, *Jan. 25-27*, 1993.

[31] P. E. Hart, “The Condensed Nearest Neighbor Rule,” in *IEEE Transactions on Information Theory 18*, vol. 14, no. 3, pp. 515-516, May 1968.

[32] R. Marsan, “Weka for Android,” [Online]. Available: https://github.com/rjmarsan/Weka-for-Android .

[33] R. Weber, H.-J. Schek and S. Blott, “A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces,” in *Proceedings of the 24th International Conference on Very Large Data Bases*, pp. 194-205, New York, 1998.

[34] T. Liu, A. W. Moore, A. G. Gray and K. Yang, “An Investigation of Practical Approximate Nearest Neighbor Algorithms,” in *Neural Information Processing Systems*, pp. 825-832, Vancouver, 2004.

[35] W. Li *et al.*, “Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement,” *eprint arXiv:1610.02455 [cs.DB]*, pp. 1-26, 2016.

[36] B. Naidan, L. Boytsov and E. Nyberg, “Permutation search methods are efficient, yet faster search is possible,” in *Proceedings of the VLDB Endowment*, pp. 1618-1629, Hawaii, 2015.

[37] L. Shi-guang and W. Yin-wei, “Fast nearest neighbor searching based on improved VP-tree,” *Pattern Recognition Letters,* vol. 60, no. C, pp. 8-15, 2015.

[38] H. Lu, A. B. Brush, B. Priyantha, A. K. Karlson and J. Liu, “SpeakerSense: Energy Efficient Unobtrusive Speaker Identification on Mobile Phones,” LNCS vol. 6696, pp. 188-205, San Francisco, 2011.

[39] N. D. Lane *et al.*, “BeWell: A Smartphone Application to Monitor, Model and Promote Wellbeing,” in *Intl. ICST Conf. on Pervasive Computing Technologies for Healthcare*, Dublin, IEEE press, 2011.